PIC 40A

Lecture 19: PHP Form handling, session variables

How does a browser communicate with a program on a server?

By submitting an HTTP request to the server (possibly via an XHTML form).

Such a request is made by a certain method. Each HTTP request method is different. There are several different possible requests, but we will concentrate on the two most frequently used ones.

HTTP request methods

GET

- Most common
- For simple document requests and small forms submission

POST

- For submitting large forms to the server to be processed

HTTP method GET

- Browsers can pass parameters to a server script via a request URI.

Example:
`http://www.pic.ucla.edu/~v/serverscript.php?x=1&y=2`

The URL of the target PHP script is:
`http://www.pic.ucla.edu/~v/serverscript.php`

The query string is:
x=1&y=2

- A form can send data this way, but you do not need a form to use this method.

Query Strings

- appear after a question mark `?` in a URL.
- consist of `name=value` pairs separated by ampersands (&).
- `name` is the `name` attribute of a widget or control of the XHTML form
- spaces in value can be replaced by + signs
- Widget values in a query string can have special characters encoded as a % followed by a 2 digit hex ASCII code representing the special character. For more detail, consult :

`http://www.permadi.com/tutorial/urlEncoding/`

GET vs. POST

GET method:

- Values (query string) are encoded directly into URI.
- Requests are cached by the browser, server, or proxy.
- Appropriate if the amount of data to send is small, and there are no side effects on the server.

POST method:

- Values encoded in a separate part of the HTTP request (query string is not visible in URL as it is in GET).
- Requests cannot be cached (each request is independent and matters).

Submitting an HTTP request using a form

- Click the submit button of the XHTML form to send form data to the script or JavaScript function that will process the form.
- The action attribute is a URI or a JavaScript function call.
- The method attribute specifies the HTTP request method.

```
<form action="http://www.ucla.edu/some.php" method="post">

<!-- Form stuff -->

</form>
```

Processing Form Data

- How does a PHP script get information from a client?
- How does a PHP script get information from the server it is running on?
- How does PHP save information from a session with a client?

Answer: Using PHP superglobal arrays

Superglobal Arrays

Superglobals are built-in variables that are always available in all scopes. There is no need to do `global $variable;` to access them within functions.

`$_SERVER`
–stores data about the currently running server.
`$_ENV`
–stores data about the current client's environment.
`$_GET`
–stores data sent to the server using HTTP method GET.
`$_POST`
–stores data sent to the server using HTTP method POST.
`$_COOKIE`
–stores data contained in cookies on the client's computer.
`$_SESSION`
–used by PHP to stores data pertaining to a the server's session with a client.

Form example using get

```
<form action="../PHP/calculator.php" method="get">
    <fieldset>
        <label for="x">x: </label>
        <input type="text" name="x" id="x" /><br/>
        <label for="y">y:</label>
        <input type="text" name="y" id="y" /><br/><br/>
        <input type="submit" value="Calculate Sum" /><br/><br/>
        <input type="reset" />
    </fieldset>
</form>
```

PHP Script calculator.php

```php
<?php

// Get the form data using form field names
// as keys for superglobalarray $_GET.

$x = $_GET['x'];
$y = $_GET['y'];

// Process form data

print("$x" . " + " . "$y = " . ($x + $y) . "<br />");

?>
```

Example using post

```
<form action="../PHP/calculator.php" method="post">
    <fieldset>
        <label for="x">x: </label>
        <input type="text" name="x" id="x" /><br/>
        <label for="y">y:</label>
        <input type="text" name="y" id="y" /><br/><br/>
        <input type="submit" value="Calculate Sum" /><br/><br/>
        <input type="reset" />
    </fieldset>
</form>
```

Processing post data

```php
<?php

// Get the form data using form field names
// as keys for superglobalarray $_POST.

$x = $_POST['x'];
$y = $_POST['y'];

// Process form data

print("$x" . " + " . "$y = " . ($x + $y) . "<br />");

?>
```

Processing form data

Often when we process form data we have to check what data was actually submitted through the form.

Common method is illustrated in the following example:

```
$from = (isset($_POST["from"]))?$_POST["from"]:"";
```

Here we are checking if `$_post` array has value for the key called `from`. If it does we simply access the key and assign the value to a PHP variable that we call `$from`. If there is no such key/value we assign the default value of empty string ("") to the `$from` variable.

We use the common C++ syntax: `(bool expression) ? do this : do that`

Where `do this` is done if bool expression is true and `do that` is done if it is false.

email example

See email example on examples page.

Some notes on PHP mail function:

bool **mail** (string `$to` , string `$subject` , string `$message` [, string `$additional_headers` [, string `$additional_parameters`]])

additional_headers are optional.
String to be inserted at the end of the email header.
This is typically used to add extra headers (From, Cc, and Bcc).

Check PHP.net for additional details on this function.

Session

- Session is the time span during which a browser interacts with a particular server.
- It begins when a browser connects to a server.
- Ends when the connection is terminated or the browser connects to a different server.

PHP Session Tracking

You can create a unique session ID for your session by calling the function `session_start()`.

Subsequent calls to `session_start()` retrieves the `$_SESSION` superglobal array.

`$_SESSION` array contains key-value pairs that were created by the script during the session.

Session folder

Create your own session save folder:

```
public_html/sessions
```

Then set the permissions to the folder so everyone can read and write to it.

Now include the line in your PHP that sets the new session save path.

```
session_save_path('/net/walnut/h1/grad/virtanen
/public_html/sessions');
```

Using session to authenticate

One of the common situations is that a person will have to log onto the website to gain access.

Once in the website they are free to move among many pages. If the user has already logged in he should not have to re-login when he goes to the next page.

We can keep track of the user and whether the user has logged in or not through the session variable.

Example

```
<?php

// At this point user has already entered login information through a form.

//This script has been called and users login and password have been compared against a database.

//The password has been found to be correct and we will now grant the user access.
```

Example continued

```
//We now record that the user has logged in by storing his
//login at the session variable.

session_start();

$_SESSION['login'] = $login;

//Now we might direct the user to the main part of the
website.

header("location: member-index.php");
```

Example continued

We will know whether the user is logged in or not by the presence or absence of login in the session variable.

If a variable login exists in the session, then the user has been logged in and authenticated.

```
session_start();

if(!isset($_SESSION['login']) {
        header("location: access-
denied.php");
    exit();
}
```

If the login is set then the script will keep running and

Example continued

We will know whether the user is logged in or not by the presence or absence of login in the session variable.

If a variable login exists in the session, then the user has been logged in and authenticated.

```
session_start();

if(!isset($_SESSION['login']) {
        header("location: access-denied.php");
    exit();
}
```

If the login is set then the script will keep running and generate rest of the page.

Example continued

To logout the user we can just unset his login in the session variable.

```
unset($_SESSION['login']);
```

How it works

When `session_start()` is called, PHP will automatically look for a session id key in the `$_COOKIE`.

If session_start() is being called the first time, then no session id exists.

A session id will be created; it looks something like:
`af48de0c4d61b0a4f49ed8c08d1e8dad`

A cookie is sent to the client's computer that looks like:
`PHPSESSID=af48de0c4d61b0a4f49ed8c08d1e8dad`

How it works continued

Recall that cookie created on one domain and path e.g. mydomain.com/mypath will be available to all web pages that reside in the same domain and path.

When we go from one PHP page to another the cookies that are created are available to all the other pages in the same domain and path.

So as we move to another PHP page, $_COOKIE on that page will contain the cookie:

```
PHPSESSID=af48de0c4d61b0a4f49ed8c08d1e8dad
```

How it works continued

So on subsequent calls to session_start() $_COOKIE already contains the session id and so PHP knows that we do not need to create a new session id.

When session_start() is first called a super global array $_SESSION is created.

PHP script may then use the $_SESSION array to write any data it wishes.

On subsequent calls to session_start() the $_SESSION array is retrieved and it contains all the previously stored data.